Course Introduction
and
C++ Basics

2017 Fall
Euiseong Seo
(euiseong@skku.edu)
Overview

- What this course is about
- Who teaches this course
- Why you have to take this course
- What you will learn in this course
- What you will earn in this course
- How to succeed in this course
What This Course is About

- C++ Programming
  - C++ Syntax
  - Object-Oriented Concept
  - Basic Programming Skills
Administrative Information

- **Course Code**
  - GEDB 030

- **Class Hour**
  - Tuesday: 09:00 ~ 10:15
  - Thursday: 10:30 ~ 11:45

- **Lecture Room**
  - #22410 (located on 4F of Engineering Bldg. I)

- **Programming Lab Location**
  - #22306 (located on 3F of Engineering Bldg. I)
**Textbook**

- **Absolute C++ (6th Edition)**
  - Walter Savitch
  - 2015 Pearson
Reference

- Programming Principles and Practice using C++ (2nd Edition)
  - Bjarne Stroustrup
  - 2015 Addison Wesley
Course Components

- Lectures
  - Concepts
  - Backgrounds

- Lab Sessions
  - Programming two simple programs every week
  - Brief explanation of solutions
Course Web Page

- http://csl.skku.edu/GEDB030F17
- Check the web site regularly
- Class material and other information will be posted
Grading

- Proportion of Activities
  - Participation 10%
  - Exams 50% (Midterm + Final)
  - Lab Sessions 40%

- If you miss any exam, you will fail
- No late attendance is allowed
- Up to four absences will be tolerated
Ethical Code

- No academic misconduct will be tolerated
  - Zero-tolerance policy
  - One who is found guilty will be kicked out of my class immediately
Lecturer

- Euiseong Seo
  - Associate professor, Software Dept.
  - E-Mail: euiseong (at) skku.edu
  - Office: #85564
  - Phone: (031) 299-4953
Learning Objectives

- Introduction to C++
  - Origins, Object-Oriented Programming, Terms
- Variables, Expressions, and Assignment Statements
- Console Input/Output
- Program Style
- Libraries and Namespaces
Introduction to C++

- **C++ Origins**
  - Low-level languages
    - Machine, assembly
  - High-level languages
    - C, C++, ADA, COBOL, FORTRAN
  - Object-Oriented-Programming in C++

- **C++ Terminology**
  - *Programs and functions*
  - Basic Input/Output (I/O) with cin and cout
Display 1.1

A Sample C++ Program

```cpp
#include <iostream>
using namespace std;

int main() {
    int numberOfLanguages;

    cout << "Hello reader.\n"
     << "Welcome to C++.\n";

    cout << "How many programming languages have you used? ";
    cin >> numberOfLanguages;

    if (numberOfLanguages < 1)
        cout << "Read the preface. You may prefer\n"
         << "a more elementary book by the same author.\n";
    else
        cout << "Enjoy the book.\n";

    return 0;
}
```
**Sample Dialogue 1**

Hello reader.
Welcome to C++.
How many programming languages have you used? 0  
User types in 0 on the keyboard.
Read the preface. You may prefer a more elementary book by the same author.

**Sample Dialogue 2**

Hello reader.
Welcome to C++.
How many programming languages have you used? 1  
User types in 1 on the keyboard.
Enjoy the book
### C++ Variables

#### C++ Identifiers
- Keywords/reserved words vs. Identifiers
- Case-sensitivity and validity of identifiers
- Meaningful names!

#### Variables
- A memory location to store data for a program
- Must declare all data before use in program
## Simple Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Memory Used</th>
<th>Size Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>short</strong> (also called short int)</td>
<td>2 bytes</td>
<td>(-32,768) to (32,767)</td>
<td>Not applicable</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>4 bytes</td>
<td>(-2,147,483,648) to (2,147,483,647)</td>
<td>Not applicable</td>
</tr>
<tr>
<td><strong>long</strong> (also called long int)</td>
<td>4 bytes</td>
<td>(-2,147,483,648) to (2,147,483,647)</td>
<td>Not applicable</td>
</tr>
<tr>
<td><strong>float</strong></td>
<td>4 bytes</td>
<td>approximately (10^{-38}) to (10^{38})</td>
<td>7 digits</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>8 bytes</td>
<td>approximately (10^{-308}) to (10^{308})</td>
<td>15 digits</td>
</tr>
</tbody>
</table>
### Data Types: Display 1.2  Simple Types (2 of 2)

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>long double</td>
<td>10 bytes</td>
<td>approximately $10^{-4932}$ to $10^{4932}$</td>
<td>19 digits</td>
</tr>
<tr>
<td>char</td>
<td>1 byte</td>
<td>All ASCII characters</td>
<td>Not applicable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Can also be used as an integer type, although we do not recommend doing so.)</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>1 byte</td>
<td>true, false</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. **Precision** refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types **float**, **double**, and **long double** are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.
Assigning Data

- Initializing data in declaration statement
  - Results "undefined" if you don’t!
    - int myValue = 0;

- Assigning data during execution
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
    - Example:
      distance = rate * time;
      Lvalue: "distance"
      Rvalue: "rate * time"
## Assigning Data: Shorthand Notations

<table>
<thead>
<tr>
<th>EXAMPLE</th>
<th>EQUIVALENT TO</th>
</tr>
</thead>
<tbody>
<tr>
<td>count += 2;</td>
<td>count = count + 2;</td>
</tr>
<tr>
<td>total -= discount;</td>
<td>total = total - discount;</td>
</tr>
<tr>
<td>bonus *= 2;</td>
<td>bonus = bonus * 2;</td>
</tr>
<tr>
<td>time /= rushFactor;</td>
<td>time = time/rushFactor;</td>
</tr>
<tr>
<td>change %= 100;</td>
<td>change = change % 100;</td>
</tr>
<tr>
<td>amount *= cnt1 + cnt2;</td>
<td>amount = amount * (cnt1 + cnt2);</td>
</tr>
</tbody>
</table>
Data Assignment Rules

Compatibility of Data Assignments

- Type mismatches
  - General Rule: Cannot place value of one type into variable of another type

- `intVar = 2.99; // 2 is assigned to intVar!`
  - Only integer part "fits", so that’s all that goes
  - Called "implicit" or "automatic type conversion"

- Literals
  - 2, 5.75, "Z", "Hello World"
  - Considered "constants": can’t change in program
Literal Data

- **Literals**
  - Examples:
    - 2 // Literal constant int
    - 5.75 // Literal constant double
    - "Z" // Literal constant char
    - "Hello World" // Literal constant string

- Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!
Escape Sequences

- "Extend" character set
- Backslash, `\` preceding a character
  - Instructs compiler: a special "escape character" is coming
  - Following character treated as "escape sequence char"
  - Display 1.3 next slide
### Some Escape Sequences

<table>
<thead>
<tr>
<th>SEQUENCE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\n</code></td>
<td>New line</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>Carriage return (Positions the cursor at the start of the current line. You are not likely to use this very much.)</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>(Horizontal) Tab (Advances the cursor to the next tab stop.)</td>
</tr>
<tr>
<td><code>\a</code></td>
<td>Alert (Sounds the alert noise, typically a bell.)</td>
</tr>
<tr>
<td><code>\</code></td>
<td>Backslash (Allows you to place a backslash in a quoted expression.)</td>
</tr>
</tbody>
</table>
### Display 1.4

**Some Escape Sequences (2 of 2)**

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\'</code></td>
<td>Single quote (Mostly used to place a single quote inside single quotes.)</td>
</tr>
<tr>
<td><code>&quot;&quot;</code></td>
<td>Double quote (Mostly used to place a double quote inside a quoted string.)</td>
</tr>
</tbody>
</table>

The following are not as commonly used, but we include them for completeness:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\v</code></td>
<td>Vertical tab</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>Backspace</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>Form feed</td>
</tr>
<tr>
<td><code>\?</code></td>
<td>Question mark</td>
</tr>
</tbody>
</table>


## Constants

### Naming your constants

- Literal constants are "OK", but provide little meaning
  - e.g., seeing 24 in a pgm, tells nothing about what it represents

### Use named constants instead

- Meaningful name to represent data
  ```
  const int NUMBER_OF_STUDENTS = 24;
  ```
  - Called a "declared constant" or "named constant"
  - Now use it’s name wherever needed in program
  - Added benefit: changes to value result in one fix
Standard Arithmetic Operators

- Precedence rules – standard rules

```cpp
#include <iostream>
using namespace std;

int main()
{
    const double RATE = 6.9;
    double deposit;

    cout << "Enter the amount of your deposit $";
    cin >> deposit;
```
Arithmetic Operators: Display 1.5  Named Constant (2 of 2)

```
10 double newBalance;
11 newBalance = deposit + deposit*(RATE/100);
12 cout << "In one year, that deposit will grow to\n"
13     << "$" << newBalance << " an amount worth waiting for.\n"
14     return 0;
15 }
```

**Sample Dialogue**

Enter the amount of your deposit $100
In one year, that deposit will grow to $106.9 an amount worth waiting for.
Type Casting

- **Casting for Variables**
  - `static_cast<double>`\(\text{intVar}\)
  - Explicitly "casts" or "converts" \(\text{intVar}\) to double type
    - Result of conversion is then used
    - Example expression:
      ```cpp
doubleVar = static_cast<double>\(\text{intVar1} / \text{intVar2}\);
```
    - Casting forces double-precision division to take place among two integer variables!
Type Casting

- Two types
  - Implicit—also called "Automatic"
    - Done FOR you, automatically
      17 / 5.5
      This expression causes an "implicit type cast" to take place, casting the 17 \(\rightarrow\) 17.0
  - Explicit type conversion
    - Programmer specifies conversion with cast operator
      (double)17 / 5.5
      Same expression as above, using explicit cast
      (double)myInt / myDouble
      More typical use; cast operator on variable
Shorthand Operators

- **Increment & Decrement Operators**
  - Just short-hand notation
  - Increment operator, `++`
    `intVar++;` is equivalent to `intVar = intVar + 1;`
  - Decrement operator, `--`
    `intVar--;` is equivalent to `intVar = intVar – 1;`
Shorthand Operators: Two Options

- **Post-Increment**
  \[ \text{intVar}++ \]
  - Uses current value of variable, THEN increments it

- **Pre-Increment**
  \[ ++\text{intVar} \]
  - Increments variable first, THEN uses new value

- "Use" is defined as whatever "context" variable is currently in

- No difference if "alone" in statement:
  \[ \text{intVar}++; \text{ and } ++\text{intVar}; \rightarrow \text{identical result} \]
Post-Increment in Action

- Post-Increment in Expressions:
  ```
  int n = 2, valueProduced;
  valueProduced = 2 * (n++);
  cout << valueProduced << endl;
  cout << n << endl;
  ```
  - This code segment produces the output:
    4
    3
  - Since post-increment was used
**Pre-Increment in Action**

- Now using Pre-increment:
  ```
  int n = 2, valueProduced;
  valueProduced = 2 * (++n);
  cout << valueProduced << endl;
  cout << n << endl;
  ```

  - This code segment produces the output:
    6
    3
  - Because pre-increment was used
Console Input/Output

- I/O objects cin, cout, cerr
- Defined in the C++ library called <iostream>
- Must have these lines (called preprocessor directives) near start of file:
  - `#include <iostream>`
  - using namespace std;
  - Tells C++ to use appropriate library so we can use the I/O objects cin, cout, cerr
### Console Output

- **What can be outputted?**
  - Any data can be outputted to display screen
    - Variables
    - Constants
    - Literals
    - Expressions (which can include all of above)
  - `cout << numberOfGames << " games played.";
    2 values are outputted:
    "value" of variable numberOfGames, literal string " games played."

- **Cascading: multiple values in one cout**
Separating Lines of Output

- **New lines in output**
  - Recall: "\n" is escape sequence for the char "newline"

- **A second method: object endl**

- **Examples:**
  
  ```cpp
  cout << "Hello World\n";
  // Sends string "Hello World" to display, & escape sequence "\n", skipping to next line
  
  cout << "Hello World" << endl;
  // Same result as above
  ```
String type

- C++ has a data type of “string” to store sequences of characters
  - Not a primitive data type; distinction will be made later
  - Must add `#include <string>` at the top of the program
  - The “+” operator on strings concatenates two strings together
  - `cin >> str` where `str` is a string only reads up to the first whitespace character
Display 1.5  Using cin and cout with a string (part 1 of 2)

```cpp
1    // Program to demonstrate cin and cout with strings
2    #include <iostream>
3    #include <string>  // Needed to access the string class.
4    using namespace std;
5    int main( )
6    {
7        string dogName;
8        int actualAge;
9        int humanAge;

10       cout << "How many years old is your dog?" << endl;
11       cin >> actualAge;
12       humanAge = actualAge * 7;

13       cout << "What is your dog's name?" << endl;
14       cin >> dogName;

15       cout << dogName << "'s age is approximately " <<
16       "equivalent to a " << humanAge << " year old human."
17       << endl;

18       return 0;
19    }
```
Display 1.5 Using `cin` and `cout` with a string (part 2 of 2)

Sample Dialogue 1

How many years old is your dog?
5
What is your dog's name?
Rex
Rex's age is approximately equivalent to a 35 year old human.

Sample Dialogue 2

How many years old is your dog?
10
What is your dog's name?
Mr. Bojangles
“Bojangles” is not read into `dogName` because `cin` stops input at the space.
Mr.'s age is approximately equivalent to a 70 year old human.
Formatting Output

- Formatting numeric values for output
  - Values may not display as you’d expect!
    
    ```cpp
    cout << "The price is "$ << price << endl;
    ```
    - If `price` (declared double) has value 78.5, you might get:
      - The price is $78.500000
      - The price is $78.5

- We must explicitly tell C++ how to output numbers in our programs!
Formatting Numbers

- "Magic Formula" to force decimal sizes:
  ```cpp
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- These stmts force all future cout’ed values:
  - To have exactly two digits after the decimal place
  - Example:
    ```cpp
cout << "The price is $" << price << endl;
    ```
    - Now results in the following:
      The price is $78.50

- Can modify precision "as you go" as well!
Error Output

- **Output with cerr**
  - cerr works same as cout
  - Provides mechanism for distinguishing between regular output and error output

- **Re-direct output streams**
  - Most systems allow cout and cerr to be "redirected" to other devices
    - e.g., line printer, output file, error console, etc.
Input Using cin

- **cin** for input, **cout** for output

**Differences:**
- ">>" (extraction operator) points opposite
  - Think of it as "pointing toward where the data goes"
- Object name "cin" used instead of "cout"
- No literals allowed for cin
  - Must input "to a variable"

- **cin >> num;**
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num
Prompting for Input: cin and cout

- Always "prompt" user for input
  `cout << "Enter number of dragons: "; cin >> numOfDragons;`
  - Note no "\n" in cout. Prompt "waits" on same line for keyboard input as follows:

    Enter number of dragons: ___

    - Underscore above denotes where keyboard entry is made

- Every cin should have cout prompt
  - Maximizes user-friendly input/output
Program Style

- **Bottom-line:** Make programs easy to read and modify

- **Comments, two methods:**
  - `//` Two slashes indicate entire line is to be ignored
  - `/*` Delimiters indicates everything between is ignored `*/`
  - Both methods commonly used

- **Identifier naming**
  - `ALL_CAPS` for constants
  - `lowerToUpper` for variables
  - Most important: MEANINGFUL NAMES!
Summary

- C++ is case-sensitive
- Use meaningful names
  - For variables and constants
- Variables must be declared before use
  - Should also be initialized
- Use care in numeric manipulation
  - Precision, parentheses, order of operations
- #include C++ libraries as needed
Summary

- **Object cout**
  - Used for console output

- **Object cin**
  - Used for console input

- **Object cerr**
  - Used for error messages

- **Use comments to aid understanding of your program**
  - Do not overcomment